#01 criando o projeto com nest

Comando nest new nomeDoProjeto, remova tudo de teste e tbm do package.json de test

```
    "rxjs": "^7.8.1"
  },
  "devDependencies": {
    "@nestjs/cli": "^10.0.0",
    "@nestjs/schematics": "^10.0.0",
    "@nestjs/testing": "^10.0.0",
    "@types/express": "^4.17.17",
    "@types/jest": "^29.5.2",
    "@types/node": "^20.3.1",
    "source-map-support": "^0.5.21",
    "ts-node": "^10.9.1",
    "tsconfig-paths": "^4.2.0",
    "typescript": "^5.1.3"
  }
}
```

#02 Módulos, serviço e controllers

Controllers é a porta de entrada para nossa aplicação via http, é tudo que recebe requisição http.

Decorators é um função que adiciona comportamento em algo.

Módulo é o arquivo que junta tudo,.

Services pode ser qualquer coisa, tudo que não recebe requisição http é provider

#03 configurando ESLint e Prettier pnpm i eslint @rocketseat/eslint-config -D o comando é pnpm run lint

app.module.ts U .eslintrc.json U

.eslintrc.json > ...

```
{
  "extends": "@rocketseat/eslint-config/node",
  "rules": {
    "no-useless-constructor": "off"
  }
}
```

.eslintignore U

.eslintignore

```
node_modules
dist
```

#04 setup Docker composse

```yaml
docker-compose.yml
version: '3.8'

services:
  postgres:
    container_name: nest-clean-postgres
    image: postgres
    ports:
      - 5432:5432
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: docker
      POSTGRES_DB: nest-clean
      PGDATA: /data/postgres
    volumes:
      - ./data/pg:/data/postgres
```

Rodar o comando Docker-compose up -d

#05 setup do prisma pnpm prisma -D e pnpm i @prisma/cliente depois rodar o comando pnmp prisma init

```
.gitignore U
.gitignore
!.vscode/extensions.json

# Env
.env

# Docker
data
```

```
model User {
  id       String @id @default(uuid())
  name     String
  email    String @unique
  password String

  questions Question[]

  @@map("users")
}

model Question {
  id        String    @id @default(uuid())
  title     String
  slug      String    @unique
  content   String
  createdAt DateTime  @default(now()) @map("created_at")
  updatedAt DateTime? @updatedAt
  authorId  String    @map("author_id")

  author User @relation(fields: [authorId], references: [id])

  @@map("questions")
}
```

rodar o comando para gerar as migrations pnpm prisma migrate dev

```
schema.prisma U   .env   docker-compose.yml U
.env
DATABASE_URL="postgresql://postgres:docker@localhost:5432/nest-clean?schema=public"
```

Abrir no navegador pnpm prisma studio

#06 Criando serviço do prisma

```python
import { Injectable, OnModuleDestroy, OnModuleInit } from '@nestjs/common'
import { PrismaClient } from '@prisma/client'

@Injectable()
export class PrismaService extends PrismaClient implements OnModuleInit, OnModuleDestroy {
  public client: PrismaClient

  constructor() {
    super({
      log: ['warn', 'error'] //faz um log do warn e error
    }) //chama o construtor da classe
  }

  onModuleInit() { //chama quando for instanciado
    return this.$connect() //conecta com prisma
  }

  onModuleDestroy() { //chama quando for destruído
    return this.$disconnect() //desconecta do prisma caso caia
  }
}
```

```python
@Module({
  controllers: [AppController],
  providers: [AppService, PrismaService],
})
export class AppModule {}
```

```python
import { Controller, Get, Post } from '@nestjs/common'
import { AppService } from './app.service'
import { PrismaService } from './prisma/prisma.service'

@Controller()
export class AppController {
  constructor(
      private appService: AppService,
      private prisma: PrismaService,
    ) {}

  @Get()
  getHello(): string {
    return this.appService.getHello()
  }

  @Post('/hello')
  async store() {
    return await this.prisma.user.findMany()
  }
}
```

#07 Controller de criação de conta

```
  "skipLibCheck": true,
  "strict": true,
  "strictNullChecks": true,
  "noImplicitAny": false,
  "strictBindCallApply": false,
  "forceConsistentCasingInFileNames": false,
```

```
import { Body, ConflictException, Controller, HttpCode, Post } from '@nestjs/common'
import { PrismaService } from '../prisma/prisma.service'

@Controller('/accounts')
export class CreateAccountController {
  constructor(private prisma: PrismaService) {} //chama o construtor

  @Post()
  @HttpCode(201) //força o retorno 201
  async handle(@Body() body: any) { //vem do corpo e salva na var body
    const { name, email, password } = body //pega de dentro do body

    const userWithSameEmail = await this.prisma.user.findUnique({
      where: {
        email
      }
    })

    if (userWithSameEmail) {
      throw new ConflictException("User with same e-mail address already exists.")
    }

    await this.prisma.user.create({
      data: {
        name, email, password
      }
    })
  }
}
```

#08 gerando hash de senha pnpm i bcryptjs e o pnpm i @types/bcryptjs no controller de user

```
      throw new ConflictException("User with same e-mail address already exists.")
    }

    const hashedPassword = await hash(password, 8)

    await this.prisma.user.create({
      data: {
        name,
        email,
        password: hashedPassword
      }
    })
  }
}
```

#09 criando pipe de validação do zod pnpm i zod, pipes são middlewares interceptadores

```
//cria o schema do zod
const createAccountBodySchema = z.object({
  name: z.string(),
  email: z.string().email(),
  password: z.string(),
})

type CreateAccountBodySchema = z.infer<typeof createAccountBodySchema>

@Controller('/accounts')
export class CreateAccountController {
  constructor(private prisma: PrismaService) {} //chama o construtor

  @Post()
  @HttpCode(201) //força o retorno 201
  async handle(@Body() body: CreateAccountBodySchema) { //vem do corpo e salva na var body
    const { name, email, password } = createAccountBodySchema.parse(body) //pega de dentro do body, validando com zod

    const userWithSameEmail = await this.prisma.user.findUnique({
      where: {
```

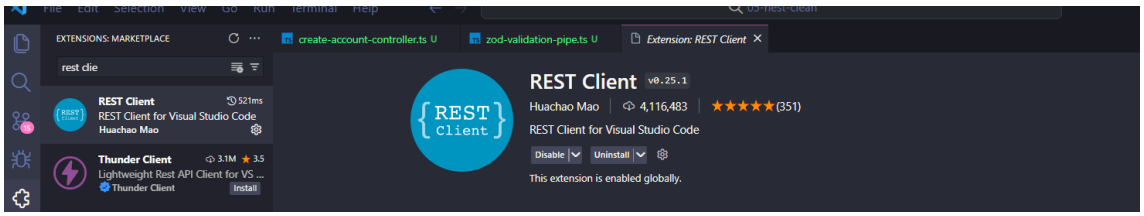
```
import { PipeTransform, ArgumentMetadata, BadRequestException } from '@nestjs/common'
import { ZodSchema } from 'zod'

export class ZodValidationPipe implements PipeTransform {
  constructor(private schema: ZodSchema) {}

  transform(value: unknown) {
    try {
      const parsedValue = this.schema.parse(value);
      return parsedValue;
    } catch (error) {
      throw new BadRequestException('Validation failed');
    }
  }
}
```

```
@Post()
@HttpCode(201) //força o retorno 201
@UsePipes(new ZodValidationPipe(createAccountBodySchema)) //usando o pipe do zod
async handle(@Body() body: CreateAccountBodySchema) { //vem do corpo e salva na var body
  const { name, email, password } = body //pega de dentro do body, validando com zod
```

Instalar pnpm i zod-validation-error

```
src > pipes > zod-validation-pipe.ts > ...
1  import { PipeTransform, BadRequestException } from '@nestjs/common'
2  import { ZodError, ZodSchema  } from 'zod'
3  import { fromZodError } from 'zod-validation-error'
4
5  export class ZodValidationPipe implements PipeTransform {
6    constructor(private schema: ZodSchema) {}
7
8    transform(value: unknown) {
9      try {
10       const parsedValue = this.schema.parse(value);
11       return parsedValue;
12     } catch (error) {
13       if (error instanceof ZodError) {
14         throw new BadRequestException({
15           message: 'Validation failed',
16           statusCode: 400,
17           errors: fromZodError(error)
18         }); //forma o error de forma mais visual
19       }
20
21       throw new BadRequestException('Validation failed');
22     }
23   }
24 }
```

#10 extensão rest cliente no vscode



Separar as requisições por ###

#11 usando configmodule no nest.js pnpm i @nestjs/config

```
EXPLORER
OPEN EDITORS
05-NEST-CLEAN
  data
  dist
  node_modules
  prisma
  src
    controllers
    pipes
    prisma
    app.module.ts U
    main.ts U
  test
  .env
  .gitignore U
  client.http U
  docker-compose.yml U
  env.ts U
```

```
env.ts U
env.ts > Env
import { z } from "zod";

export const envSchema = z.object({
  DATABASE_URL: z.string().url(),
  PORT: z.coerce.number().optional().default(3333),
})

export type Env = z.infer<typeof envSchema>
```

```
env.ts U   app.module.ts U   main.ts U
src > app.module.ts > ...
import { Module } from '@nestjs/common'
import { PrismaService } from './prisma/prisma.service'
import { CreateAccountController } from './controllers/create-account-controller'
import { ConfigModule } from '@nestjs/config'
import { envSchema } from '../env'

@Module({
  imports: [ConfigModule.forRoot({
    validate: (env) => envSchema.parse(env),
    isGlobal: true
  })],
  controllers: [CreateAccountController],
  providers: [PrismaService],
})
export class AppModule {}
```

```
src > main.ts > ...
import { NestFactory } from '@nestjs/core'
import { AppModule } from './app.module'
import { ConfigService } from '@nestjs/config'
import { Env } from '../env'

async function bootstrap() {
  const app = await NestFactory.create(AppModule)

  const configService: ConfigService<Env, true> = app.get(ConfigService) //pega uma dependência da aplicação
  const port = configService.get('PORT', { infer: true })

  await app.listen(port)
}
bootstrap()
```

#12 configurando autenticação jwt instalar pnpm i @nestjs/passport @nestjs/jwt

```
auth
  auth.module.ts U
  controllers
  pipes
  prisma
```

```
    validate: (env) => envSchema.parse(env),
    isGlobal: true
  }),
  AuthModule, //funciona para todo o app
],
```

```
auth.module.ts U   env.ts U
src > env.ts > Env
import { z } from "zod";

export const envSchema = z.object({
  DATABASE_URL: z.string().url(),
  JWT_SECRET: z.string(),
  PORT: z.coerce.number().optional().default(3333),
})
```

```typescript
import { Module } from '@nestjs/common'
import { PassportModule } from '@nestjs/passport'
import { JwtModule } from '@nestjs/jwt'
import { ConfigService } from '@nestjs/config'
import { Env } from '../env'

@Module({
  imports: [
    PassportModule,
    JwtModule.registerAsync({
      inject: [ConfigService], //lista de serviço injetado quando registro esse módulo
      useFactory(config: ConfigService<Env, true>) {
        const secret = config.get('JWT_SECRET', { infer: true })

        return {
          secret,
        }
      },
    })
  ]
})
export class AuthModule {}
```

#13 gerando token jwt o comando para gerar é “openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:2048” e depois “openssl rsa -pubout -in private_key.pem -out public_key.pem”

https://travistidwell.com/jsencrypt/demo/

certutil -encode private_key.pem private_key-base64.txt

certutil -encode public_key.pem public_key-base64.txt

```typescript
import { z } from "zod";

export const envSchema = z.object({
  DATABASE_URL: z.string().url(),
  JWT_PRIVATE_KEY: z.string(),
  JWT_PUBLIC_KEY: z.string(),
  PORT: z.coerce.number().optional().default(3333),
})

export type Env = z.infer<typeof envSchema>
```

```typescript
@Module({
  imports: [
    PassportModule,
    JwtModule.registerAsync({
      inject: [ConfigService], //lista de serviço injetado quando registro esse módulo
      global: true,
      useFactory(config: ConfigService<Env, true>) {
        const privateKey = config.get('JWT_PRIVATE_KEY', { infer: true })
        const publicKey = config.get('JWT_PUBLIC_KEY', { infer: true })

        return {
          signOptions: { algorithm: 'RS256' },
          privateKey: Buffer.from(privateKey, 'base64'),
          publicKey: Buffer.from(publicKey, 'base64'),
        }
      },
    })
  ]
})
export class AuthModule {}
```

```
import { Controller, Post } from '@nestjs/common'
import { JwtService } from '@nestjs/jwt'

@Controller('/sessions')
export class AuthenticateController {
  constructor(private jwt: JwtService) {}

  @Post()
  async handle() {
    const token = this.jwt.sign({ sub: 'user-id' })

    return token
  }
}
```

```
],
controllers: [CreateAccountController, AuthenticateController],
providers: [PrismaService],
```

#14 controller de autenticação

```
//cria o schema do zod
const authenticateBodySchema = z.object({
  email: z.string().email(),
  password: z.string(),
})

type AuthenticateBodySchema = z.infer<typeof authenticateBodySchema>

@Controller('/sessions')
export class AuthenticateController {
  constructor(
      private prisma: PrismaService,
      private jwt: JwtService
    ) {}

  @Post()
  @UsePipes(new ZodValidationPipe(authenticateBodySchema)) //usando o pipe do zod
  async handle(@Body() body: AuthenticateBodySchema) {
    const { email, password} = body

    const user = await this.prisma.user.findUnique({
      where: { email }
    })

    if (!user) {
      throw new UnauthorizedException("User credentials do not match.")
    }

    const isPasswordValid = await compare(password, user.password)

    if (!isPasswordValid) {
      throw new UnauthorizedException("User credentials do not match.")
    }

    const accessToken = this.jwt.sign({ sub: user.id })

    return {
      access_token: accessToken
    }
  }
}
```

#15 protegendo rotas com guards instalar o pnpm i passport-jwt e pnpm i @types/passport-jwt

```
import { JwtStrategy } from './jwt.strategy'

@Module({
  imports: [ ...
  ],
  providers: [JwtStrategy]
})
export class AuthModule {}
```

```typescript
import { ConfigService } from '@nestjs/config'
import { PassportStrategy } from '@nestjs/passport'
import { ExtractJwt, Strategy } from 'passport-jwt'
import { Env } from '../env'
import { z } from 'zod'
import { Injectable } from '@nestjs/common'

const tokenSchema = z.object({
  sub: z.string().uuid()
})

type TokenSchema = z.infer<typeof tokenSchema>

@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy) {
  constructor(config: ConfigService<Env, true>) {
    const publicKey = config.get('JWT_PUBLIC_KEY', { infer: true })

    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(), //pega o token do header
      secretOrKey: Buffer.from(publicKey, 'base64'),
      algorithms: ['RS256']
    }) //chama o construtor da classe strategypassport
  }

  async validate(payload: TokenSchema) {
    return tokenSchema.parse(payload)
  }
}
```

```typescript
import { AuthGuard } from "@nestjs/passport";

export class JwtAuthGuard extends AuthGuard('jwt') {}
```

```typescript
import { Controller, Post, UseGuards, UsePipes } from '@nestjs/common'
import { JwtAuthGuard } from '../auth/jwt-auth.guard'

@Controller('/questions')
@UseGuards(JwtAuthGuard)
export class CreateQuestionController {
  constructor() {}

  @Post()
  async handle() {
    return 'ok'
  }
}
```

#16 criando decorator de autenticação

```typescript
import { ExecutionContext, createParamDecorator } from '@nestjs/common'
import { UserPayLoad } from './jwt.strategy'

export const CurrentUser = createParamDecorator(
  (_: never, context: ExecutionContext) => {
    const request = context.switchToHttp().getRequest()

    return request.user as UserPayLoad
  }
)

  async handle(@CurrentUser() user: UserPayLoad) {
    console.log(user) // tem os dados do usuário
    return 'ok'
  }
}
```

#17 controller de criação de pergunta

```
create-question-controller.ts U ×   create-account-controller.ts U
src > controllers > create-question-controller.ts > CreateQuestionController > handle

const createQuestionBodySchema = z.object({
  title: z.string(),
  content: z.string(),
})

type CreateQuestionBodySchema = z.infer<typeof createQuestionBodySchema>

@Controller('/questions')
@UseGuards(JwtAuthGuard)
export class CreateQuestionController {
  constructor(
    private prisma: PrismaService
  ) {}

  @Post()
  async handle(
      @Body(new ZodValidationPipe(createQuestionBodySchema)) body: CreateQuestionBodySchema,
      @CurrentUser() user: UserPayLoad
    ) {
    console.log(user) // tem os dados do usuário
    const { content, title } = body
    const { sub: userId } = user

    const slug = this.convertToSlug(title)

    await this.prisma.question.create({
      data: {
        authorId: userId,
        title,
        content,
        slug
      }
    })
  }
```

```
    }
  })
}

private convertToSlug(title: string): string {
  return title
    .toLowerCase()
    .normalize('NFD')
    Test Regex...
    .replace(/[\u0300-\u036f]/g, '')
    Test Regex...
    .replace(/[^\w\s-]/g, '')
    Test Regex...
    .replace(/\s+/g, '-')
}
}
```

#18 controller de listagem de pergunta

```
export class ZodValidationPipe implements PipeTransform {
  constructor(private schema: ZodSchema) {}

  transform(value: unknown) {
    try {
      const parsedValue = this.schema.parse(value);
      return parsedValue; //permite fazer a validação e transformação
    } catch (error) {
      if (error instanceof ZodError) {
```

```
pipes
  zod-validation-pipe.ts U
prisma
app.module.ts U
env.ts U
main.ts U
```

```
    CreateAccountController,
    AuthenticateController,
    CreateQuestionController,
    FetchRecentQuestionsController
  ],
```

```
import { ZodValidationPipe } from '../pipes/zod-validation-pipe'
import { z } from 'zod'

const pageQueryParamSchema = z.string().optional().default('1').transform(Number).pipe(
  z.number().min(1)
)

const queryValidationPipe = new ZodValidationPipe(pageQueryParamSchema)

type PageQueryParamSchema = z.infer<typeof pageQueryParamSchema>

@Controller('/questions')
@UseGuards(JwtAuthGuard)
export class FetchRecentQuestionsController {
  constructor(
    private prisma: PrismaService
  ) {}

  @Get()
  async handle(@Query('page', queryValidationPipe) page: PageQueryParamSchema){
    const perPage = 1

    const questions = await this.prisma.question.findMany({
      take: perPage,
      skip: (page - 1) * perPage, //pula registro necessário para eu mostra registro da próxima página
      orderBy: {
        createdAt: 'desc'
      }
    })

    return { questions }
  }
}
```

#19 configurando vitest com swc instalar pnpm i vitest unplugin-swc @swc/core @vitest/coverage-v8 -D

```
vitest.config.ts
import swc from 'unplugin-swc'
import { defineConfig } from 'vitest/config'

export default defineConfig({
  test: {
    globals: true,
    root: './',
  },
  plugins: [
    // This is required to build the test files with SWC
    swc.vite({
      // Explicitly set the module type to avoid inheriting this value from a `.swcrc` config file
      module: { type: 'es6' },
    }),
  ],
});
```

Instalar tbm o pnpm i vite-tsconfig-paths -D

```
pnpm-lock.yaml
private_key.pem
public_key.pem
tsconfig.build.json
tsconfig.json

  "forceConsistentCasingInFileNames": false,
  "noFallthroughCasesInSwitch": false,
  "paths": {
    "@/*": ["./src/*"]
  }
```

```typescript
import swc from 'unplugin-swc'
import { defineConfig } from 'vitest/config'
import tsConfigPaths from 'vite-tsconfig-paths'

export default defineConfig({
  test: {
    globals: true,
    root: './',
  },
  plugins: [
    tsConfigPaths(),
    // This is required to build the test files with SWC
    swc.vite({
      // Explicitly set the module type to avoid inheriting this value from a `.swcrc` config file
      module: { type: 'es6' },
    }),
  ],
});
```

Para os controller cria teste E2E, peagr apenas os test e2e

```typescript
import swc from 'unplugin-swc'
import { defineConfig } from 'vitest/config'
import tsConfigPaths from 'vite-tsconfig-paths'

export default defineConfig({
  test: {
    include: ['**/*.e2e-spec.ts'],
    globals: true,
    root: './',
  },
  plugins: [
    tsConfigPaths(),
    // This is required to build the test files with SWC
    swc.vite({
      // Explicitly set the module type to avoid inheriting this value from a `.swcrc` config file
      module: { type: 'es6' },
    }),
  ],
});
```

```json
    "start:debug": "nest start --debug --watch",
    "start:prod": "node dist/main",
    "lint": "eslint \"{src,apps,libs,test}/**/*.ts\" --fix",
    "test": "vitest run",
    "test:watch": "vitest",
    "test:cov": "vitest run --coverage",
    "test:debug": "vitest --inspect-brk --inspect --logHeapUsage --threads=false",
    "test:e2e": "vitest run --config ./vitest.config.e2e.ts"
  },
```

```typescript
  "types": [
    "vitest/globals"
  ]
}
}
```

#20 banco de dados isolando nos test

```
    globals: true,
    root: './',
    setupFiles: ['./test/setup-e2e.ts'] //é chamado antes de cada arquivo de teste
  },
  plugins: [
    tsConfigPaths(),
    // This is required to build the test files with SWC
    swc.vite({
      // Explicitly set the module type to avoid inheriting this value from a `.swcrc` config file
      module: { type: 'es6' },
    }),
  ],
});
```

Instalar o pnpm i dotenv -D

```
import { PrismaClient } from '@prisma/client'
import { execSync } from 'child_process'
import { randomUUID } from 'crypto'
import 'dotenv/config'

const prisma = new PrismaClient()

function generateUniqueDatabaseURL(schemaId: string) {
  if (!process.env.DATABASE_URL) {
    throw new Error('Please provider a DATABASE_URL environment variable.')
  }

  const url = new URL(process.env.DATABASE_URL)

  url.searchParams.set('schema', schemaId)

  return url.toString()
}

const schemaId = randomUUID();

beforeAll(async () => {
  const databaseURL = generateUniqueDatabaseURL(schemaId)

  process.env.DATABASE_URL = databaseURL //recebe a nova schema da url

  execSync('pnpm prisma migrate deploy') //comando de rodar no terminal as migrations
})

afterAll(async () => {
  await prisma.$executeRawUnsafe(`DROP SCHEMA IF EXISTS "${schemaId}" CASCADE`)
  await prisma.$disconnect()
})
```

#21 testes e2e de usuário instalar pnpm i supertest -D e também pnpm i @types/supertest -D

```
import { INestApplication } from '@nestjs/common'
import { Test } from '@nestjs/testing'
import { AppModule } from '../app.module'
import request from 'supertest'
import { PrismaService } from '../prisma/prisma.service'

describe('Create account (E2E)', () => {
  let app: INestApplication
  let prisma: PrismaService

  beforeAll(async () => {
    const moduleRef = await Test.createTestingModule({
      imports: [AppModule],
    }).compile();

    app = moduleRef.createNestApplication()

    prisma = moduleRef.get(PrismaService) //para ter acesso ao prisma

    await app.init();
  });
```

```
  test('[POST] /accounts', async () => {
    const response = await request(app.getHttpServer()).post('/accounts').send({
      name: 'José Pereira',
      email: 'jose@email.com',
      password: '123456'
    })

    expect(response.statusCode).toBe(201)

    const userOnDatabase = await prisma.user.findUnique({
      where: {
        email: 'jose@email.com'
      }
    })

    expect(userOnDatabase).toBeTruthy() //valida que crio no banco
  })
})
```

```
  test('[POST] /sessions', async () => {
    await prisma.user.create({
      data: {
        name: 'José Pereira',
        email: 'jose@email.com',
        password: await hash('123456', 8)
      }
    })

    const response = await request(app.getHttpServer()).post('/sessions').send({
      email: 'jose@email.com',
      password: '123456'
    })

    expect(response.statusCode).toBe(201)
    expect(response.body).toEqual({
      access_token: expect.any(String)
    })
  })
})
```

#22 testes e2e de perguntas

```
import { INestApplication } from '@nestjs/common'
import { Test } from '@nestjs/testing'
import { AppModule } from '../app.module'
import request from 'supertest'
import { PrismaService } from '../prisma/prisma.service'
import { JwtService } from '@nestjs/jwt'

describe('Create question (E2E)', () => {
  let app: INestApplication
  let prisma: PrismaService
  let jwt: JwtService

  beforeAll(async () => {
    const moduleRef = await Test.createTestingModule({
      imports: [AppModule],
    }).compile();

    app = moduleRef.createNestApplication()

    prisma = moduleRef.get(PrismaService) //para ter acesso ao prisma
    jwt = moduleRef.get(JwtService);

    await app.init();
  });
```

```
  test('[POST] /questions', async () => {
    const user = await prisma.user.create({
      data: {
        name: 'José Pereira',
        email: 'jose@email.com',
        password: '123456'
      }
    })

    const access_token = jwt.sign({ sub: user.id })

    const response = await request(app.getHttpServer())
      .post('/questions')
      .set('Authorization', `Bearer ${access_token}`)
      .send({
        title: 'New questions',
        content: 'Questions content',
      })

    expect(response.statusCode).toBe(201)

    const questionOnDatabase = await prisma.question.findFirst({
      where: {
        title: 'New questions',
      }
    })

    expect(questionOnDatabase).toBeTruthy() //valida que crio no banco
  })
})
```

```
  test('[GET] /questions', async () => {
    const user = await prisma.user.create({
      data: {
        name: 'José Pereira',
        email: 'jose@email.com',
        password: '123456'
      }
    })

    const access_token = jwt.sign({ sub: user.id })

    await prisma.question.createMany({
      data: [
        {
          title: 'Question 01',
          slug: 'question-01',
          content: 'Question content',
          authorId: user.id
        },
        {
          title: 'Question 02',
          slug: 'question-02',
          content: 'Question content',
          authorId: user.id
        },
        {
          title: 'Question 03',
          slug: 'question-03',
          content: 'Question content',
          authorId: user.id
        },
      ]
    })
```

```
59
60      const response = await request(app.getHttpServer())
61        .get('/questions')
62        .set('Authorization', `Bearer ${access_token}`)
63        .send()
64
65      expect(response.statusCode).toBe(200)
66      expect(response.body).toEqual({
67        questions: [
68          expect.objectContaining({ title: 'Question 01' }),
69          expect.objectContaining({ title: 'Question 02' }),
70          expect.objectContaining({ title: 'Question 03' }),
71        ]
72      })
73    })
74  })
```

Módulo 02